J Grid Computing (2016) 14:359–378
DOI 10.1007/s10723-015-9355-6

Storage-aware Algorithms for Scheduling of Workflow Ensembles in Clouds

Piotr Bryk · Maciej Malawski · Gideon Juve · Ewa Deelman

Received: 8 June 2015 / Accepted: 29 October 2015 / Published online: 17 November 2015
© The Author(s) 2015. This article is published with open access at Springerlink.com

Abstract This paper focuses on data-intensive workflows and addresses the problem of scheduling workflow ensembles under cost and deadline constraints in Infrastructure as a Service (IaaS) clouds. Previous research in this area ignores file transfers between workflow tasks, which, as we show, often have a large impact on workflow ensemble execution. In this paper we propose and implement a simulation model for handling file transfers between tasks, featuring the ability to dynamically calculate bandwidth and supporting a configurable number of replicas, thus allowing us to simulate various levels of congestion. The resulting model is capable of representing a wide range of storage systems available on clouds: from in-memory caches (such as memcached), to distributed file systems (such as NFS servers) and cloud storage (such as Amazon S3 or Google Cloud Storage). We observe that file transfers may have a significant impact on ensemble execution; for some applications up to 90 % of the execution time is spent on file transfers. Next, we propose and evaluate a novel scheduling algorithm that minimizes the number of transfers by taking advantage of data caching and file locality. We find that for data-intensive applications it performs better than other scheduling algorithms. Additionally, we modify the original scheduling algorithms to effectively operate in environments where file transfers take non-zero time.

Keywords Workflow ensembles · Scheduling algorithms · Cloud computing · Cloud storage

P. Bryk
Google Poland, Warsaw Financial Center, Emilii Plater 53, Warszawa, Poland
e-mail: bryk@google.com

M. Malawski (✉)
AGH, Department of Computer Science, Al. Mickiewicza 30, Kraków, Poland
e-mail: malawski@agh.edu.pl

G. Juve · E. Deelman
USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA, USA

G. Juve
e-mail: gideon@isi.edu

E. Deelman
e-mail: deelman@isi.edu

1 Introduction

Today, workflows are frequently used to model large-scale distributed scientific applications. They facilitate the expression of multi-step task workloads in a manner which is easy to understand, debug, and maintain. By using scientific workflows multiple researchers can collaborate on designing a single distributed application. This is because workflows are arranged as

directed acyclic graphs (DAGs), where each node is a standalone task and edges represent dependencies between tasks. These dependencies are typically input/output files that need to be transferred between tasks. With the rise in popularity of workflows in the scientific community, specialized execution management systems have emerged to provide dedicated execution environments. For example, Pegasus [21], which is used in a number of scientific domains, e.g. astronomy and bioinformatics, is a system that can execute workflows on desktops, clusters, grids, or clouds. Such execution is a non-trivial task, especially on clouds, where resource provisioning and deprovisioning, cost accounting, and resource setup must be taken into account. Additionally, large-scale computations are often composed of several interrelated workflows grouped into *ensembles* consisting of workflows that have a similar structure, but may differ in their input data, number of tasks, and individual task sizes.

In this paper we focus on data-intensive workflows. For example, Montage [30] is a software toolkit for constructing science-grade mosaics in the Flexible Image Transport System format. It composes multiple astronomical images into a larger mosaic. An example Montage workflow that consists of 10,429 tasks, requires 4.93 CPU hours to execute, reads 146.01 GB of input data, and writes 49.93 GB of output data [32]. CyberShake [15] workflows, which are used to generate seismic hazard maps, are similarly data-intensive. A typical CyberShake run requires 9,192.45 CPU hours to execute and reads 217 TB of data [32]. In cloud environments, where observed sustained global storage throughput is on the order of 10-20 MiB/s [14], data transfer times are not negligible and may take a significant portion of the total workflow execution time. Several studies confirm that the transfers may also comprise a significant amount of workflow execution cost [33, 42, 53]. Dedicated workflow-aware file systems have even been proposed to address this issue [19].

Although data transfers play an important role in the performance of workflows on clouds, most existing research work on scheduling does not adequately address this issue. Workflow scheduling algorithms often assume that data is transferred directly between execution units. This is the case for Heterogeneous Earliest Finish Time (HEFT) and other similar heuristics [6, 12, 48]. However, in clouds global storage systems are the most popular means of file transfers [33]. Examples of such storage systems include object storage (e.g. Google Cloud Storage [3]), shared Network File System (NFS) folders, parallel file systems (e.g. Lustre) or even in-memory data stores (e.g. memcached [23]). The goal of this work is to develop models of such storage systems and use them to evaluate scheduling algorithms.

This paper builds upon the execution model, simulation procedure, and scheduling algorithms we proposed in [39]. The paper, addresses the problem of workflow ensemble scheduling in IaaS clouds with budget and deadline constraints. In this work static (offline) and dynamic (online) scheduling and resource provisioning algorithms are proposed and analyzed with regard to various environment parameters, such as task runtime variance or virtual machine (VM) provisioning delay. However, there is an assumption that the file transfer time between tasks is either negligible or included in task runtimes. While this assumption may be correct for some types of workflows, for data-intensive workflows it may lead to incorrect or overly optimistic schedules.

In this paper we explore the area of workflow ensemble scheduling algorithms that are aware of the underlying storage architecture and can consider data transfers between tasks when scheduling ensembles. The motivation for this work is to determine how data transfers influence ensemble execution under budget and deadline constraints and how execution systems should handle data-intensive ensembles. The main contributions of this paper are fourfold. (1) We develop and implement a global storage model for our Cloud Workflow Simulator, to be used for transferring files between tasks. The model features the ability to dynamically calculate bandwidth and supports a configurable number of replicas, thus allowing us to simulate various levels of congestion in the system. (2) Based on this model we modify the original scheduling algorithms to take file transfers into account. (3) We introduce a new scheduling algorithms that takes advantage of file caching to speed up ensemble execution. (4) Finally, we discuss how various storage systems affect execution of workflow ensembles on clouds.

The paper is organized as follows. Section 2 presents related work. Section 3 describes the problem and introduces the aforementioned storage model. In Section 4 we introduce alterations to the algorithms

developed in our previous work. Section 5 contains a description of the evaluation procedure we employed and Section 6 discusses the results of our evaluation. Finally, Section 7 outlines general conclusions and explores possibilities for future work.

2 Related Work

Many scientific applications are represented as workflows of tasks. As a result, workflow scheduling has become an important and popular topic of research. Workflow scheduling algorithms have been widely studied and there are numerous works on algorithms for scheduling single workflows onto generic execution units. This includes algorithms like HEFT [48], Predict Earliest Finish Time (PEFT) [6], Lookahead [12] and many others. While these algorithms provide good results for single workflows, they are not directly applicable to the execution environment we consider in this research (IaaS clouds), where compute resources can be provisioned and deprovisioned on demand. Moreover, they also do not consider storage architectures, which is one of the main contributions of our work.

There are also efforts which focus on the problem of workflow scheduling with only one constraint, for example, cost, deadline or storage space. Chen et al. [17] address the problem of scheduling large workflows onto execution sites while staying within storage constraints. The algorithm they propose minimizes runtime and does not consider resource cost. The authors of [44] introduce an algorithm for static cost minimization and deadline constrained scheduling. Their solution is very relevant, because they target realistic clouds with heterogeneous VMs and provisioning/deprovisioning delays. They also model file transfers between tasks as peer-to-peer messages. This is different from our approach where we employ global storage system for file transfers. We believe that the global storage model is more applicable to IaaS clouds where providers offer storage services that are cheap and reliable. In [29] the authors consider inter-cloud data transfer aspects for scheduling business processes on hybrid clouds, with the objective of minimizing cost. The Hybrid Cloud Optimized Cost (HCOC) scheduling algorithm [13] minimizes execution cost given deadline constraints. It supports clouds by provisioning resources and can handle multi-core VMs. This algorithm and its variations are different from our work because we consider scheduling of workflow ensembles with two constraints. There is other research that focuses on scheduling ensembles with multiple constraints [22, 47], but that execution model differs from ours, where we schedule workflows according to their priorities.

The general problem of transferring data between workflow tasks and storage has also been the subject of research. Most of the time scheduling algorithms take generic communication cost into consideration [48]. Such algorithms do not directly apply to our execution model, where tasks do not communicate directly with each other, but instead stage files to and from the global storage system. There are algorithms [42] that schedule workflows while minimizing storage usage, e.g. in [11] the goal is to minimize the storage footprint in the case of limited space on grid execution sites, or to minimize storage costs. This is accomplished by introducing cleanup jobs into the workflow. This is an interesting problem, but we do not consider it in this work. Our global cloud storage assumes no space constraints, while for the local cache we use the FIFO policy instead of cleanup jobs. Juve et al. [33] evaluate data sharing options on IaaS clouds. They use a very similar scheme for transferring files between workflow tasks, and their approach is implemented using a global storage system. However, they do not evaluate any data-aware scheduling algorithms. Nevertheless, this reinforces that our execution model is valid and used in real-world applications.

Stork [35] is a data-aware scheduler that has been developed for efficient management of data transfers and storage space, but it does not address workflows or clouds. In [51] data clustering techniques are used for distributing datasets of workflow applications between cloud storage sites. In [16] an integrated task and data placement algorithm for workflows on clouds, based on graph partitioning is derived, with the goal of minimizing data transfers. The approach to use data locality for efficient task scheduling is also widely applied to MapReduce, where various improvements over default Hadoop scheduling are proposed [27]. Bharathi et al. [10] analyze the impact of data staging strategies on workflow execution on clouds. They present decoupled, loosely-coupled and tightly-coupled models for transferring data between tasks, based on their interaction with

the workflow manager. They observe that decoupling file transfer tasks from computations can result in significant makespan reduction for data-intensive workflows. Ranganathan et al. [43] present data management techniques for computational jobs in grids. They use simulation procedures to analyze algorithms that place computational tasks on sites that already have input files for the tasks. Pereira et al. [41] propose a scheduler for data-intensive workflows in public clouds. Their model differs from ours in that they consider disks attached to VMs (such as e.g. EBS on Amazon EC2) and assume peer to peer transfers between VMs, while we assume a global cloud storage. Moreover, their scheduler based on integer linear programming (ILP) has been evaluated on small scale workflows, while we address ensembles of large-scale workflows. Chiang et al. [18] addresses the problem of scheduling VMs on physical machines to avoid interference between I/O intensive applications using shared resources of the host physical machine. They propose an interesting method of workload modeling using machine learning. Their infrastructure model assumes local disks on physical hosts or iSCSI attached volumes, which is different from our global storage model.

None of the aforementioned related work considers scheduling algorithms for workflow ensembles on IaaS clouds that optimize data transfers and include a flexible data storage model. In this research we tackle this interesting research problem.

3 Problem Description

In this section we introduce the main assumptions of our application and environment model, provide details of the proposed storage model, and define the main performance metric used for evaluation.

3.1 Execution Model

The execution model used to evaluate our scheduling and provisioning algorithms is an extension of the model proposed in [39]. We introduce additions to support modelling file storage and transfers between workflow tasks.

A cloud consists of an unlimited number of homogeneous single-core virtual machines (VMs) that can be provisioned and deprovisioned at any time. The reason for such a cloud model is the fact that studies say that there is typically one VM type that is best suited for a particular workflow application [33].

There is a delay between the time a VM provisioning (deprovisioning) request is sent and the time the VM becomes available for execution. This is to account for typical startup and shutdown delays that are present in real-world public clouds [28, 40].

VMs can execute only one task at a time. A VM not executing any task is called an idle VM. When a task is submitted to a non-idle VM, it is enqueued for execution. When a task successfully finishes execution on a VM, a queued task (if any) is dequeued and executed according to the FIFO rule. A VM is charged \$1 for each N-minute interval it has been running. Partial usage within a billing interval is rounded up. Unless specified otherwise, `N` is assumed to be 60 (one hour). Again, this assumption is made for simplicity and to reflect typical billing practices of cloud providers [31].

Target applications are ensembles of workflows (sets of workflows). Workflow tasks have runtime estimates that indicate how long it takes to execute them on a given VM type. Unlike in the original model, the estimates do not include file transfers or any other kind of communication costs. Runtime estimates can often be obtained from preliminary runs of workflows [33, 49]. Some workflow applications, like Periodograms [9], even include an analytical performance model that estimates task runtimes. To account for the fact that actual task runtime depends on the dynamic state of the system it runs on (e.g., CPU cache, disk or network latencies), and often differs from predictions, we randomize runtimes under a uniform distribution of $\pm p$ %. Workflows have integral, non-negative priorities that indicate their importance for completion. The priority system is exponential, meaning that it is better to complete a workflow with priority p than any number of workflows with lower priorities. This assumption reflects the reality of many scientific applications, for example CyberShake [26, 37], where it is better to compute a seismic hazard at the location of a single tall building than for an entire unpopulated desert area.

A task has zero or more input files that have to be staged entirely to a VM before it can start executing the task. Similarly, the task has zero or more output files, which may be used as inputs to other task or be the end result of a workflow. File names are unique within a workflow, meaning that two files with the

same names but referenced in different workflows are considered different. These assumptions are in line with real-world execution environments like Pegasus and its DAX file format for workflow representation [21]. Similar to runtimes, we assume that file sizes can be estimated on the basis of historical data. Files are assumed to be write-once, read-many (WORM), meaning that a file is never updated once it has been written.

The goal of scheduling and resource provisioning is to complete as many workflows as possible within the given deadline and budget constraints, respecting workflow priorities. A workflow is considered complete if all its jobs and file transfers have completed successfully within the budget and deadline constraints.

Our simulator has an event loop with a global timer. At the beginning of a simulation the event loop calls the initial scheduling procedures, sets a global timer to zero and then waits for incoming events. The simulation is considered completed when there are no more events in the event loop. Each event is a quartet of: type, payload, destination, and delay time t_d. When an event is sent at time t, it is received at the destination on time $t + t_d$. With such an architecture we are able to simulate file transfer times, delays and varying task runtimes.

3.2 Storage and File Transfer Model

We propose a global storage model to be used for transferring input and output files between tasks, as well as for storing the final results of the application for retrieval. Each VM has a local cache of files that were generated on the VM or transferred to the VM from the global storage system. To stage in (out) a file to (from) a VM, a request has to be sent to the global storage management system. The system then transfers the file to (from) the VM according to the dynamic state, configuration parameters, file size and file presence in the per-VM local cache. We have chosen a global storage model for file transfers instead of peer-to-peer transfers because this paradigm is widely used in grid environments in the form of simple NFS folders or shared distributed filesystems [25, 45]. What is more, Agarwal et al. show that peer-to-peer data sharing in workflow applications does not perform well on typical public clouds [5]. Cloud providers offer globally accessible APIs for storing and retrieving files, for example Google Cloud Storage. Our model supports all aforementioned storage models from a single shared disk to highly scalable distributed storage systems.

We assume that transfers between tasks always require uploading or downloading entire files. This is because most cloud storage systems are not POSIX-compliant and offer little support for reading parts of files [52]. With this assumption we additionally offload part of the burden of file transfer optimization to workflow application developers, who should design their applications so that input files are fully utilized to minimize unnecessary data transfers.

Although we acknowledge that cloud storage services are not free, we consider storage usage charges to be outside the scope of this research. This means that the amount of bytes stored and transferred does not affect the total cost of ensemble execution. The rationale behind this assumption is the fact that in many cases *all* input and output files are preserved for further analysis after workflow completion. What is more, the typical price for storing all of a workflow's files in the cloud for the duration of its execution is much smaller than the price for computing. For example, the Workflow Gallery [46] provides a sample Montage application consisting of 1000 tasks that executes in 3 hours 10 minutes and generates files with a total size of 4.2GiB. According to current Google Cloud pricing [2], it costs substantial more to rent standard VMs for the duration of computation ($0.1583 at $0.05 per hour) than to store all files for the same amount of time ($0.0001128 at $0.026 per GiB per month that consists of 730 hours). Berriman et al. [8] provide similar numbers, showing that short-term storage is much cheaper than computation.

In the model we assume that before executing any task all input files have to be fully staged in from the global storage to a VM. Staging of input files occurs in sequence, i.e. input file transfer I starts only after transfer $I - 1$ has successfully finished. This is done intentionally; in our congestion model parallel transfers would not provide any advantages for a single task, since the bandwidth of a link to a VM is a limiting factor. More importantly, for transferring files to multiple tasks, starting simultaneous transfers could delay execution of those transfers that are almost complete. If a file requested for staging in is not the output of an already completed task, it is assumed to be a workflow input file that has been prepared and

uploaded prior to execution. Task output files are handled in a similar fashion. This process is outlined in Fig. 1. A similar approach has already been employed in real-world workflow applications, such as the one described in [33].

To model the dynamic state of the global storage system we introduce the concept of *replicas* responsible for handling file operations. A replica is a storage device that contains copies of all files and fulfills file staging requests. The system is characterized by maximum read b_r and write b_w bandwidths that never change. The bandwidths are assigned to all the endpoints, so that congestion can occur on both VM and replica endpoints, and they are identical for all replicas across the system. Read and write bandwidths are independent, meaning that reading does not affect writing and is processed separately. The storage system consists of r replicas. Simple NFS folders and RAID arrays are modeled by a low number of replicas, whereas scalable distributed storage services like Google Cloud Storage or Amazon S3 are modeled by a larger number of replicas. The number of replicas remains constant throughout execution, meaning that only one storage type can be modelled at a time.

At any given point during workflow execution all currently running staging requests for a given file are handled equally by all replicas. The replicas assign a fair share of their bandwidth to each request. A file cannot be transferred faster than a given maximum bandwidth, which is identical for replicas and VMs. The bandwidth accounts for both VM limitations (e.g., network connectivity speed) and storage device characteristics (e.g., physical disk speed). The process of assigning dynamic bandwidth is outlined in Algorithm 1. Figure 2 shows sample states of the system.

Each request to the storage system is handled with a latency of l in milliseconds to account for any network and system delays. The latency parameter has the highest impact on execution of ensembles with

Algorithm 1 Dynamic bandwidth calculation in the model

Require: b_r, b_w: maximum read and write bandwidths; r: number of storage replicas
Ensure: B_r, B_w: dynamic read and write bandwidths
procedure CALCULATE BANDWIDTH(b_r, b_w, r)
  $T_r \leftarrow$ set of running read transfers
  $T_w \leftarrow$ set of running write transfers
  if $\overline{T_r} > 0$ **then**
    $B_r \leftarrow \text{MIN}(b_r r / \overline{T_r}, b_r)$
  else
    $B_r \leftarrow \emptyset$
  end if
  if $\overline{T_w} > 0$ **then**
    $B_w \leftarrow \text{MIN}(b_w r / \overline{T_w}, b_w)$
  else
    $B_w \leftarrow \emptyset$
  end if
end procedure

large numbers of small files. Recent studies confirm that distributed storage systems have non-negligible latencies, even on the order of hundreds of milliseconds [34]. We do not include additional factors such as metadata overheads, because the storage is modelled to support only writing and reading files and we believe these factors would not significantly influence the results.

Read transfer finish time t_e of a file with size s that started at time t_s can be computed by solving Equation 1, where B_r is the bandwidth assigned to the transfer changing over time.

$$s = \int_{t_s}^{t_e} B_r \, \mathrm{d}t \tag{1}$$

Because $B_r(t)$ is a discontinuous function with finite time intervals, where it is defined by constant functions, we can precisely compute the time it takes to transfer a file. The computation algorithm works as follows. When a file transfer is started, its last modification time t_m is set to the current time and the bytes remaining to be transferred, s_d, are set to the

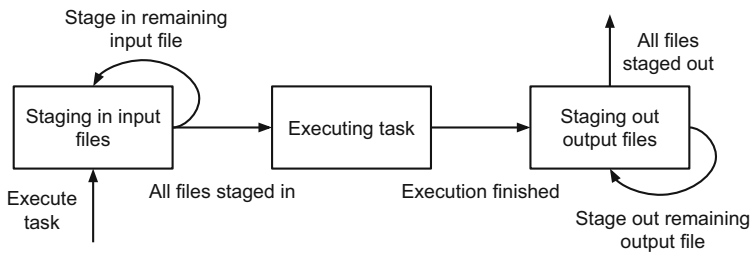

Fig. 1 State diagram of a VM that is requested to execute a task

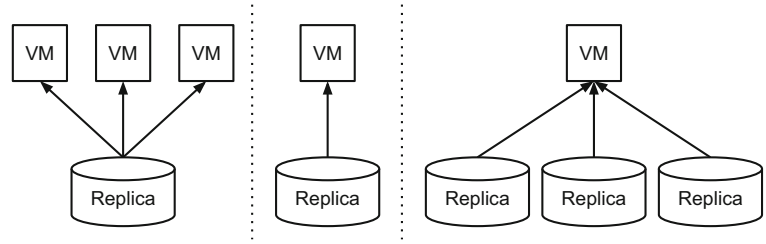

Fig. 2 VMs staging in files from storage systems with different numbers of replicas. In the left-most diagram each VM is assigned $\frac{1}{3}b_r$ bandwidth while in the central diagram only one VM is assigned b_r bandwidth. The VM in the right-most diagram is assigned a bandwidth of b_r – the bandwidth cannot be higher since the bottleneck is the VM endpoint

file size. Then, for each file transfer the s_d variable is decreased by the number of bytes transferred since the last modification time, t_m, and t_m is set to the current time. Any transfer that has zero bytes remaining is removed. The next task is to compute the lowest expected remaining transfer time, t_l, based on the current bandwidth and schedule the algorithm to be invoked again no later that t_l from the current time. The transfer update procedure is formally described in Algorithm 2 (Fig. 3).

A VM may have a local disk of size c bytes to serve as a file cache. We acknowledge that local disks are often billed separately from VMs, but for the purposes of this research we assume that the price of the disk is included in the VM cost. In the case of in-memory storage systems RAM may be used for file cache. All the input and output files of tasks executed on the VM are stored in the cache according to its policy. The files are cached in the order they were staged in to or out from the VM. We have modelled the cache using a First-In-First-Out (FIFO) policy [20]. The cache discards the least recently stored files first. When a file is required for task execution as input, and it is available in the VM cache, no staging request is issued and the file can be used immediately. The file caching system, if enabled, works passively during ensemble execution, i.e. no action is required from the scheduling and provisioning algorithms to take advantage of it.

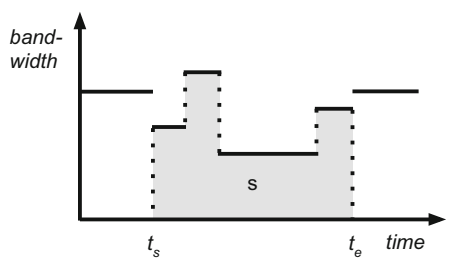

Fig. 3 Sample storage bandwidth function with file transfer start t_s, finish t_e and size s marked

Algorithm 2 Updating file transfers in the simulator

Require: B: current read (write) bandwidth; t: current time
  procedure UPDATE_TRANSFERS(B ,t)
    $G \leftarrow$ set of running read (write) transfers
    for g in G **do**
      $t_m \leftarrow$ MODIFICATION_TIME(g)
      $s_d \leftarrow$ REMAINING_BYTES(g)
      REMAINING_BYTES(g) $\leftarrow s_d - (t - t_m) * B$
      MODIFICATION_TIME(g) $\leftarrow t$
    end for
    $t_n \leftarrow$ inf ▷ Next scheduled update time
    for g in G **do**
      $s_d \leftarrow$ REMAINING_BYTES(g)
      if $s_d = 0$ **then**
        $G \leftarrow G \setminus \{g\}$ ▷ Remove finished transfer
      end if
      $t_p \leftarrow s_d / B$ ▷ Predicted remaining time
      $t_n \leftarrow$ MIN(t_n, t_p)
    end for
    SCHEDULE_UPDATE(t_n) ▷ Schedule next update no later than t_n from now
  end procedure

3.3 Performance Metric

Scheduling algorithms are evaluated using the performance metric defined in our previous paper [39]. The metric is as follows: for a given ensemble e executed under budget b and deadline d, get all successfully completed workflows and calculate the sum of their partial priority-based scores. The formal definition of the priority-based exponential score is:

$$Score(e, b, d) = \sum_{w \in Completed(e,b,d)} 2^{-Priority(w)} \quad (2)$$

The higher $Score(e, b, d)$, the better the corresponding ensemble execution is. The performance metric is defined in this way to remain consistent with our assumption that it is better to complete a workflow with priority p than any number of workflows with lower priority.

4 Algorithms

Several dynamic and static scheduling algorithms were introduced in our previous paper [39]. These algorithms are responsible for two aspects: resource provisioning (creating instances of VMs) and scheduling (assigning tasks to VMs). The algorithms developed earlier include: Dynamic Provisioning Dynamic Scheduling (DPDS), Workflow-Aware DPDS (WA-DPDS) and Static Provisioning Static Scheduling (SPSS).

In this section we first describe our modifications to the earlier algorithms. These modifications are designed to account for an execution environment in which file transfers take non-zero time. In addition, we describe a new scheduling algorithm that takes advantage of caches and file locality to improve performance. The new File Locality-Aware Scheduling Algorithm can be then combined with dynamic provisioning algorithms of DPDS and WA-DPDS. The resulting algorithms are called Dynamic Provisioning Locality-Aware Scheduling (DPLS) and Storage- and Workflow-Aware DPLS (SWA-DPLS).

4.1 Storage Aware Scheduling Algorithms

All the algorithms from [39] use a runtime prediction function $R(t)$ to estimate how long it will take to execute a task t. The function is used, for example, to calculate the critical path length in static algorithms. This function is defined as follows:

$$R(t) = UniformDistribution(Runtime(t), p) \quad (3)$$

where $Runtime(t)$ is task's runtime defined in the workflow and p is uniform distribution radius in percent. We define a new prediction function, $R_s(t)$, that includes input and output file transfer times estimations as:

$$R_s(t) = R(t) + \sum_{f \in Files(t)} T(f) \quad (4)$$

where $Files(t)$ is a set of task input and output files and $T(f)$ is an optimistic transfer time prediction function. $T(f)$ is defined as:

$$T(f) = \begin{cases} \frac{Size(f)}{b_r}, & \text{if f is an input file} \\ \frac{Size(f)}{b_w}, & \text{otherwise,} \end{cases} \quad (5)$$

where b_r and b_w are maximum read and write bandwidths as defined in Section 3.2 and $Size(f)$ is the size of the file in bytes. We call this modified prediction function optimistic because it uses the maximum available bandwidth for estimating the transfer time. We acknowledge that this may be inaccurate because of congestion effects that may occur during ensemble execution or concurrent access to the same storage device. However, this is intentionally not optimized further, because we only need a lower bound on the transfer time. The modified versions of the algorithms that use the prediction function $R_s(t)$ are called Storage-Aware DPDS (SA-DPDS), Storage- and Workflow-Aware DPDS (SWA-DPDS) and Storage-Aware SPSS (SA-SPSS).

4.2 File Locality-Aware Scheduling Algorithm

The original dynamic (online) scheduling algorithms schedule workflows onto randomly selected, idle VMs. They do not exploit information about files that are already present in the VM's local cache. As a result, when a task is submitted to run on a VM there may exist another VM where it would finish earlier by using cached data. Based on this observation, we have developed a novel dynamic scheduling algorithm that takes advantage of file locality to produce better schedules. The algorithm examines the VMs' caches at the time of task submission and chooses the VM on which the task is predicted to finish earliest, according to runtime and file transfer time estimates.

The algorithm uses a modified task runtime prediction function, based on the function in (4). The function ignores transfer time estimates for files that are already present on a VM and it adds remaining runtime estimates for all tasks that are queued

or currently running on the VM. Queued tasks are considered because they are already scheduled for execution on the VM.

$$R_{ncf}(t, vm) = R(t) + \sum_{f \in NCF(t,vm)} T(f) \quad (6)$$

$$R_{vm}(t, vm) = R_{ncf}(t, vm) + \sum_{s \in S(vm)} R_{ncf}(s, vm) - A(s), \quad (7)$$

where vm is a Virtual Machine and $NCF(t, vm)$ is the set of all input files of task t that are not present in the local file cache of vm (the set of not cached files). $S(vm)$ is a set of currently queued and running tasks on the VM and $A(s)$ is current runtime of a task s, or none if not running. Such prediction function was designed to determine at what point in time task t is predicted to finish on a possibly non-idle virtual machine vm. The predicted speedup S of a task t executing on vm is defined as the difference between the predicted runtimes from Equations 4 and 7.

$$S(t, vm) = R_s(t) - R_{vm}(t, vm). \quad (8)$$

This function allows us to estimate the speedup of running a task on a selected VM that may contain cached input files. The speedup may, of course, be negative, meaning that the task will finish earlier on an idle VM with an empty cache rather than on the selected one. This may happen for example when the selected VM executes a long-running task or its cache is empty.

The scheduling procedure is shown in Algorithm 3. At the beginning, a family of sets P_i containing ready tasks with priority i is created. A task is called ready when all its input dependant tasks have successfully completed. Root tasks of a workflow are tasks that have no input dependencies. Root tasks, by definition, are always ready and are initially added to their respective P_i sets. The algorithm operates until the deadline is reached and schedules tasks when there is at least one idle VM and there is at least one ready task. The condition concerning existence of at least one idle VM was introduced to have at least one *good* candidate VM, thereby avoiding queuing many tasks onto a single VM. Without this condition all ready tasks would be immediately scheduled onto VMs, which are (most likely) not idle, making the algorithm essentially a static one. In the task submission block, the algorithm computes the predicted speedup for each pair of: ready task with highest priority, and possibly non-idle VM. Subsequently, the pair consisting of the task and the VM with highest speedup is selected for submission. Once the task finishes, its ready children are added to the P_i sets, according to their priorities.

Algorithm 3 File Locality-Aware Dynamic Scheduling

```
Require: p_max: maximal priority in the ensemble; e: ensemble to schedule
  procedure SCHEDULE(p_max)
    for 0 ≤ i ≤ p_max do
      P_i ← ∅                          ▷ List of ready tasks with priority i
    end for
    for root task t in WORKFLOWS(e) do
      P_PRIORITY(t) ← P_PRIORITY(t) ∪ {t}
    end for
    while deadline not reached do
      IdleVMs ← set of idle VMs
      P_highest ← non-empty P_i with lowest i, or ∅ if none
      if P_highest ≠ ∅ and IdleVMs ≠ ∅ then
        VMs ← set of running VMs
        b_s ← − inf                    ▷ Highest speedup
        b_t ← null                     ▷ Task of the highest speedup
        b_vm ← null                    ▷ VM of the highest speedup
        for t in P_highest do
          for VM in VMs do
            s ← S(t, VM)               ▷ (8)
            if s > b_s then
              b_s ← s
              b_t ← t
              b_vm ← VM
            end if
          end for
        end for
        SUBMIT(b_t, b_vm)
        Update P_i sets once b_t finishes
      end if
    end while
  end procedure
```

The locality-aware scheduling algorithm ensures that ready tasks with the highest priority are submitted to VMs in the order of their predicted speedup ranking. Lower-priority tasks are always deferred when there is at least one higher-priority ready task waiting for execution. A lower-priority task submitted to an idle VM, however, may start execution earlier than a higher-priority task that was submitted to a non-idle VM. This is schematically explained in Fig. 4. Additionally, with VM caching disabled, idle VMs are always chosen for task submission, meaning that the procedure reduces to the dynamic scheduling

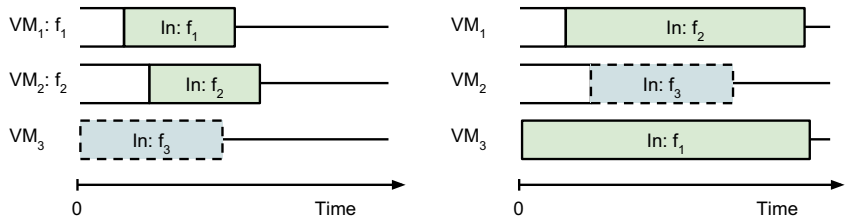

Fig. 4 Schedules produced by locality-aware algorithms with cache (*left*) and with no cache (*right*). At time 0 VM_1 and VM_2 are busy. There are also three ready tasks, the one with the dashed border has lower priority. On the left, VM_1 and VM_2 contain files f_1 and f_2 in their respective caches. Higher-priority tasks are submitted to non-idle VMs, which makes them start later than the lower-priority task. However, these tasks finish earlier than without the cache, as shown on the right

algorithm defined in [39]. Finally, the dynamic provisioning algorithms from [39] can be used together with the locality-aware scheduling procedure. The resulting algorithms are called Dynamic Provisioning Locality-Aware Scheduling (DPLS) and Storage- and Workflow-Aware DPLS (SWA-DPLS).

A summary of algorithms and their characteristics is presented for convenience in Table 1.

5 Evaluation Procedure

The algorithms were evaluated using the Cloud Workflow Simulator [1]. This simulator supports all of the assumptions that were stated in the problem description in Section 3. The simulator was extended to support the storage model introduced in this paper. New scheduling algorithms were developed and the original algorithms were extended, as described in Section 4. In this paper our intent was to focus thoroughly on simulation studies. We simulated hundreds of thousands of ensemble executions with various parameters, which would be unfeasible to run on a real testbed.

We evaluated the algorithms using ensembles consisting of synthetic workflows from the Workflow Gallery [46]. We have selected workflows representing several different classes of applications [32]. The selected applications include: CyberShake [37], a data-intensive application used by the Southern California Earthquake Center to calculate seismic hazards, Montage [30], an I/O-bound workflow used by astronomers to generate mosaics of the sky, and Epigenomics and SIPHT (sRNA Identification Protocol using High-throughput Technology) [36], two CPU-bound bioinformatics workflows.

The ensembles for each application type used in the experiments were created from randomly selected

Table 1 Summary of the algorithms

	Provisioning	Scheduling	Workflow-Aware	Storage-Aware	File Locality-Aware
SPSS	static	static	+	−	−
DPDS	dynamic	dynamic	−	−	−
WA-DPDS	dynamic	dynamic	+	−	−
SA-SPSS	static	static	+	+	−
SA-DPDS	dynamic	dynamic	−	+	−
SWA-DPDS	dynamic	dynamic	+	+	−
DPLS	dynamic	dynamic	−	+	+
SWA-DPLS	dynamic	dynamic	+	+	+

Scheduling refers to the process of assigning tasks to VMs, while provisioning is the process of creating and terminating VM instances. Workflow-aware algorithms use the information on the workflow structure in their decisions. Storage-aware algorithms take into account the estimated data transfer times when scheduling tasks. File locality-aware algorithms use the new scheduling algorithms that take the advantage of caches and file locality to improve performance

workflows from that application. Their sizes (number of tasks) were chosen according to a modified Pareto distribution described in [39]. The distribution selects large workflows (of size greater than 900 tasks) with a slightly higher-than-standard Pareto distribution probability. The priorities are assigned to workflows according to their size: the larger the workflow is, the higher priority it obtains. This strategy aims to reflect the typical scientific ensemble structure, where there are a few large and important workflows and many small workflows of lower importance. Unless specified otherwise, we used ensembles with 20 workflows.

For each experiment, the maximum and minimum viable budgets and deadlines were computed. The exact calculation procedure is described in [39]. 10 evenly chosen points in the interval between the minimum budget and the maximum budget, and 10 evenly distributed points in the interval between the minimum deadline and the maximum deadline, were used to run 100 simulations with different pairs of deadline and budget.

Unless specified otherwise, the simulations were run with task runtime estimation uncertainties of ±5%. This number was chosen to reflect the fact that real-world estimates are not perfect. The VM provisioning delay parameter was set to 120 seconds, which is typical of what can be observed in public clouds [40]. The VM deprovisioning delay was set to 60 seconds to take into account any cleanup and shutdown operations.

Table 2 summarizes the input parameters for all of the experiments to follow. The experiments model cloud environments with four different storage systems: infinitely fast storage, in-memory storage, distributed storage, and an NFS-like file system.

6 Results and Discussion

In this section we analyze the relative performance of our proposed scheduling algorithms on clouds with different storage system configurations. Additionally, we investigate how these storage systems affect the execution of ensembles.

6.1 No Storage System

In the first experiment we evaluated the performance of our algorithms in an environment where there is actually "no storage". This is equivalent to having a storage system with infinite read and write bandwidths and zero latency. In such a system all file transfers finish instantaneously. This setup allows us to compare Storage- and File Locality-Aware versions of the algorithms with their unaware counterparts and determine whether our modifications introduced any performance degradation.

The experiment consists of 100 simulations (10 budgets and 10 deadlines) for each application ensemble consisting of 50 workflows (5 applications) and each scheduling algorithm variant (7 algorithms). The experiment was repeated 10 times with different randomly-chosen ensembles and all results were aggregated into one dataset. This represents a total of 35,000 simulation runs.

Table 2 Summary of input parameters for all experiment runs

	No storage	In-memory	Distributed	NFS
Read/write bandwidth	∞	100 MiB/s	10 MiB/s	20 MiB/s
Operation latency	0	1 ms	50 ms	200 ms
Local VM cache size	0	50 GiB		
Number of replicas	∞	1, 2, 5, 10, 50	∞	5
VM provisioning time	120 s			
VM deprovisioning time	60 s			
Runtime variance	±5%			
Ensemble size	20, 50	20		
Budgets/deadlines	10			

Column headers represent storage types modeled in an experiment while row headers present input parameter names. The rationale and sources for input parameter values are described in detail in the experiments section

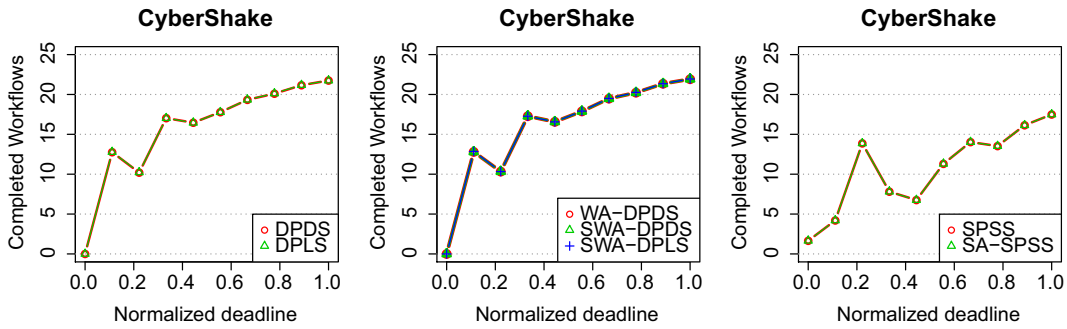

Fig. 5 Average number of completed workflows within budget and deadline constraints for scheduling algorithm families. The underlying storage system is infinitely fast

Figure 5 shows the average number of workflows completed by the normalized deadline. Deadlines are normalized to the 0-1 range for each experiment run. Only the results for CyberShake are included here because they are representative for all applications. The performance of the Storage- and File Locality-Aware scheduling algorithms was identical to that of their unaware counterparts, with very minor differences resulting from randomization. This is visible in Fig. 5, where the lines for all the algorithms overlap almost perfectly. This is consistent with our expectations, given that the storage-aware algorithms are designed to reduce to the storage-unaware procedures when the storage is infinitely fast. It is also worth not-

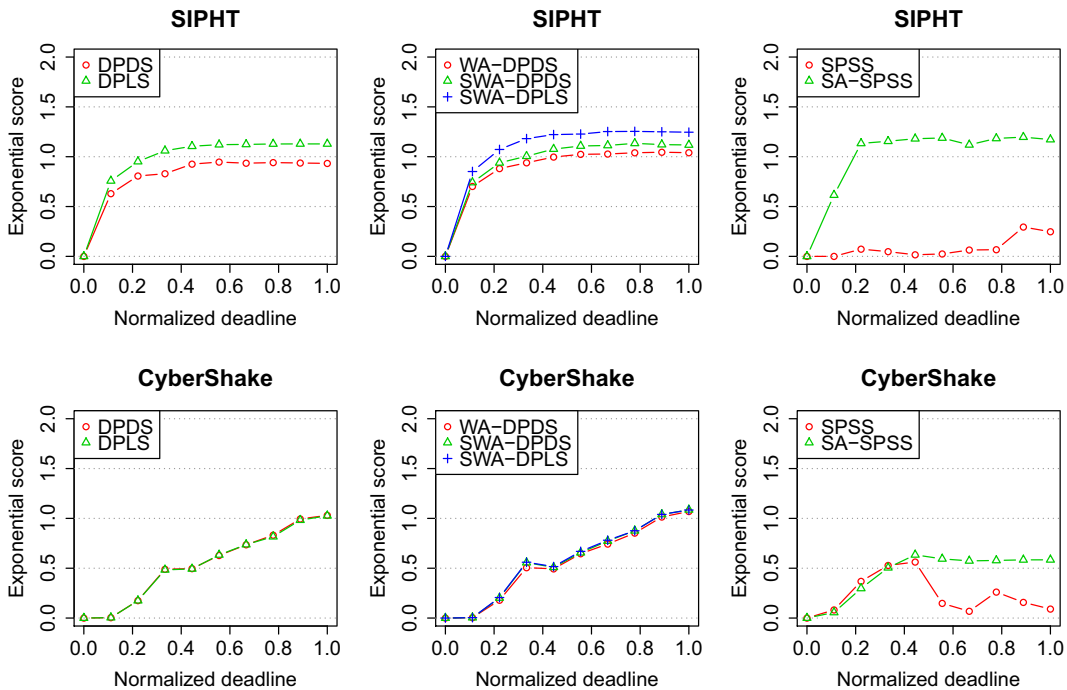

Fig. 6 Average algorithm exponential score for SIPHT and CyberShake applications running in an environment with an in-memory storage system

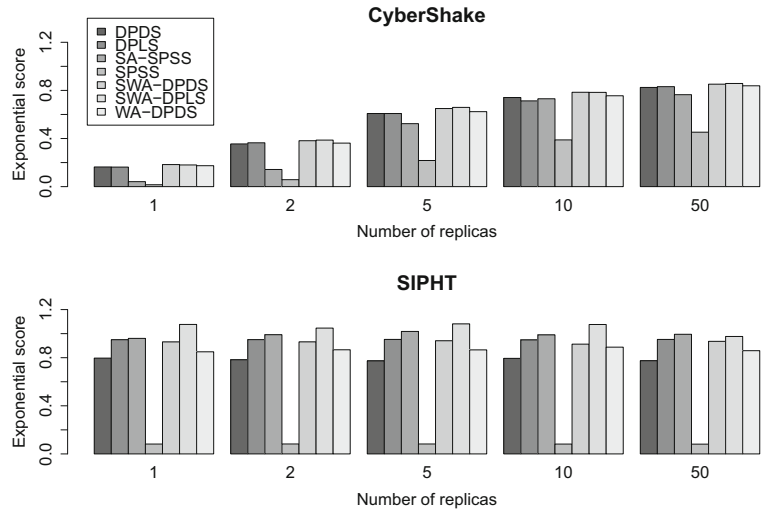

Fig. 7 Average algorithm exponential score in the function of the number of replicas of in-memory storage system

ing that the results presented here are similar to those from [39], which confirms that our experiment setup is correct.

6.2 In-memory Storage

In-memory storage systems have long been known for their high performance and throughput [24]. They are widely used in response time critical applications, e.g., in telecommunications systems. Recently, they have become popular for caching files in high-traffic web applications [50], for example using the memcached distributed cache system [23]. Such systems can be also used for storing and transferring workflow input and output files. In our experiment, the in-memory storage system is modelled with maximum read and write bandwidths of 100 MiB/s. This number reflects the approximate upper limit of throughput for the Gigabit Ethernet networks that are common in commercial clouds. Latency is set to 1ms, because memory storage systems are often simple key-value arrays that are bounded primarily by network delays. The experiments were run with 1, 2, 5, 10 and 50 replicas respectively. Again, this is to model typical in-memory storage systems that can easily scale by adding nodes that replicate data. The local VM cache size was set to 50 GiB, as this the amount of RAM that can be used for cache on high-memory VMs on public clouds (there are VMs on Google Compute Engine that provide more than 100 GiB of RAM [4].)

Figure 6 shows the results of 70,000 simulation runs (10 deadlines x 10 budgets x 4 applications x 7 algorithms x 5 replicas x 5 experiments). The Y-axis represents the average exponential score from definition 2, while the X-axis represents the normalized deadline as introduced in Section 6.1. The first observation is that the score is always zero for the minimal normalized deadline. This is expected, because deadlines and budgets are computed using estimates of task computation time only, and are set to barely allow for the execution of the smallest workflow with low safety margins. Knowing that file transfers take non-zero time, it is expected that hardly any workflow can successfully complete within the minimal deadline.

The performance of the dynamic algorithms varies slightly, depending on whether it is aware of the existence of the underlying storage or not, and on

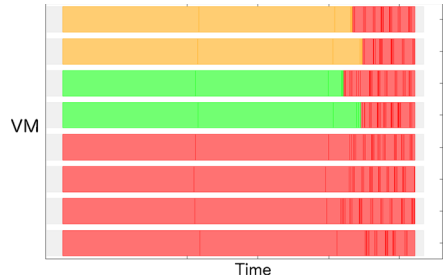

Fig. 8 Example CyberShake ensemble execution generated by the DPDS algorithm on a cloud with distributed storage system. Horizontal bars are VMs and colored ranges are task executions which include file transfers. Tasks from different workflows have separate colors. It is worth noting that initial tasks of each workflow take a long time, which is caused by large input files that need to be staged in to a VM

the type of application. This is because such algorithms are, by design, able to adapt to a changing environment and are resilient to uncertainties in tasks runtimes. The two applications shown in Fig. 6 are both data-intensive applications. For CyberShake the results are very consistent across algorithms because it contains a low number of potentially cacheable files, resulting in lower advantage for the locality-aware algorithms. Other data-intensive applications, such as SIPHT, often exhibit larger differences between the algorithms. With more lenient deadlines, DPLS performs noticeably better than DPDS, because, as the deadline extends into the future, there are more opportunities to leverage the local VM cache and thus improve performance. The family of Workflow-Aware algorithms produces similar results. SWA-DPLS outperforms other algorithms for the SIPHT application, while for CyberShake it is only slightly better. Also, SWA-DPDS is better than WA-DPDS because it is able to admit or reject workflows more accurately.

The static algorithms, on the other hand, show significant differences in performance between storage-aware and unaware variants. Figure 6 shows that for CyberShake application the performance of SA-SPSS is better than SPSS. This is even more visible for SIPHT, where SA-SPSS always produces better schedules than SPSS. Fragility with respect to runtime

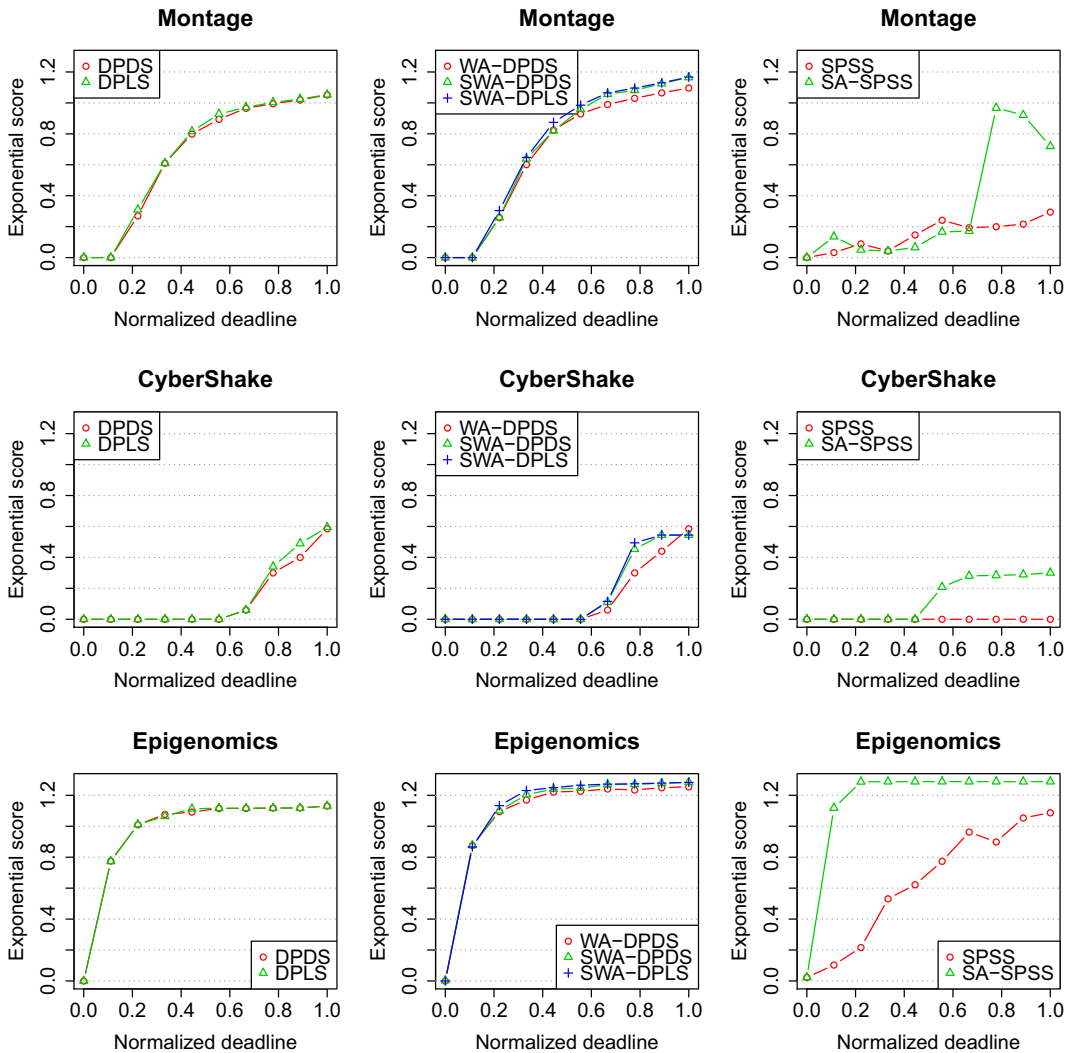

Fig. 9 Average exponential score for applications running in an environment with a distributed storage system

uncertainties is the main reason behind this behavior. Static algorithms rely on accurate task runtime estimates, and when transfer times are not included, the schedule plan degrades considerably. During the planning phase the static algorithm tries to squeeze as many workflows as possible within the budget and deadline constraints, leaving no margins of safety. Therefore, when the runtimes are underestimated (i.e. no transfers included) most workflows that would normally finish just before the deadline are considered failures. Finally, one can observe that, as the deadline for CyberShake application grows, the score for SPSS gets lower. The reason is that SPSS begins to admit the highest priority workflow (which is very large) for execution at around a normalized deadline of 0.45. However, due to underestimates in task runtimes, the execution of that workflow fails and the score becomes significantly lower.

Figure 7 shows algorithm performance as a function of the number of replicas. For SIPHT, all algorithms perform approximately the same, regardless of the number of replicas. This application exhibits low parallelism, therefore increasing the number of storage replicas affects its execution only a little. CyberShake yields the opposite result. There is high correlation between the number of replicas and average score for all algorithms. The root cause behind this effect is that CyberShake workflows start with

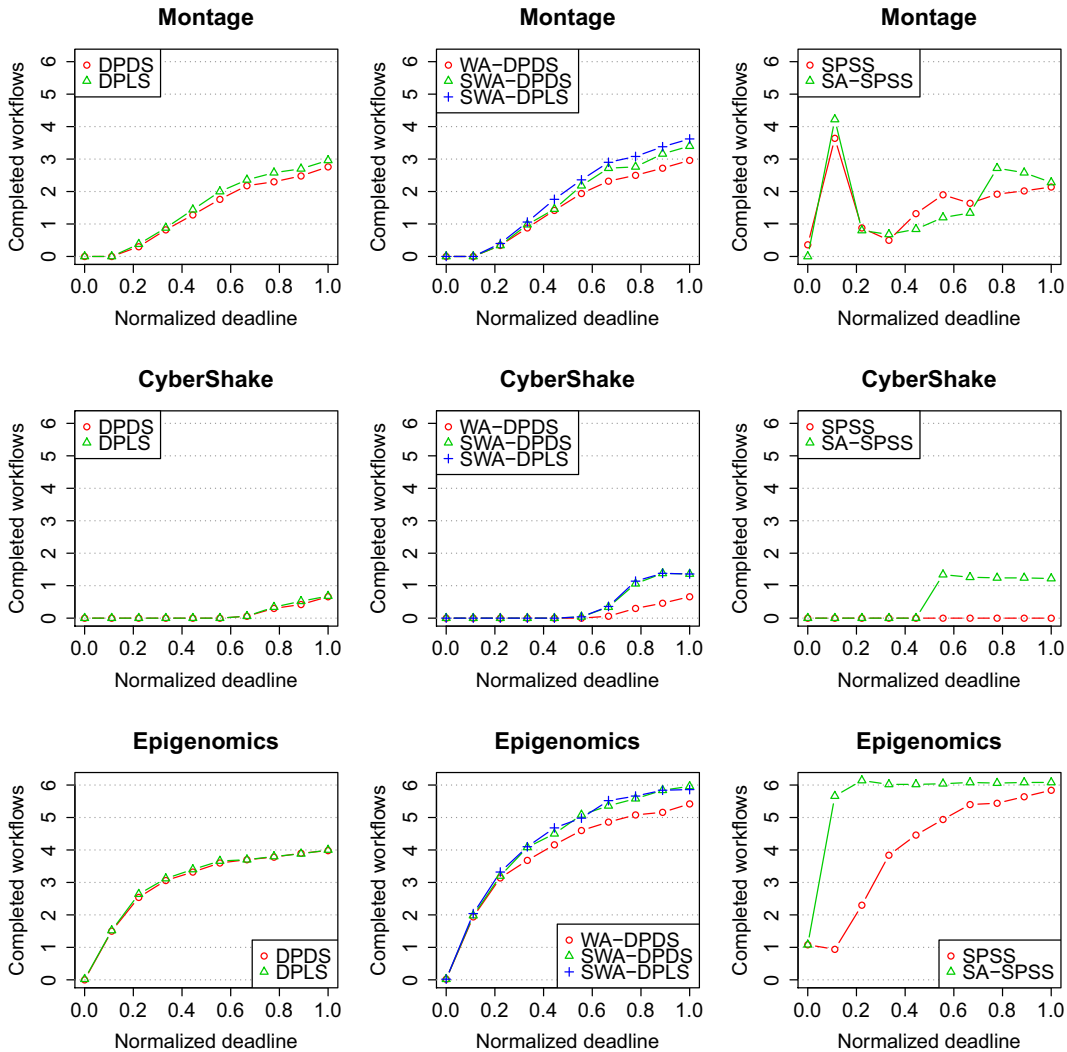

Fig. 10 Average number of completed workflows in an environment with a distributed storage system

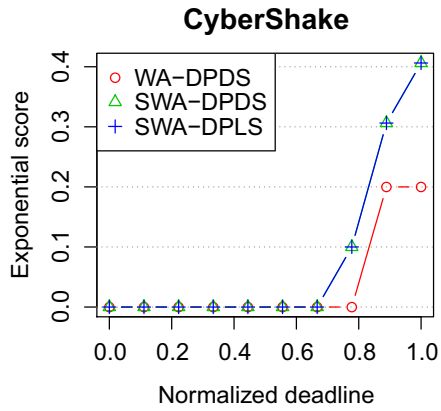

Fig. 11 Exponential score in environment with distributed storage system and disabled VM cache

tasks that require very large input files, with size in the order of 20GB. Scheduling algorithms often start many workflows at a time, which results in many parallel transfers. This situation is illustrated in Fig. 8. Finally, parallel transfers take less time when there are more replicas, which is consistent with the results.

6.3 Distributed Storage System

Distributed storage systems, such as Amazon S3, are among the most popular solutions for storing and transferring data in clouds. This is because most cloud providers offer massively scalable, managed services for storing objects. To model a distributed object storage system we have set the number of replicas to infinity. With this, we are able to simulate the behavior of popular cloud storage systems [25], where application developers only have access to a storage API and the cloud provider manages the system to scale it on demand to keep its parameters (e.g., throughput) at constant levels. We used a latency of 50ms and a local VM cache of size 50GiB. The total number of simulations run was 21,000 (10 deadlines x 10 budgets x 3 applications x 7 algorithms x 10 experiments).

Figure 9 shows the average score of the algorithm as a function of the normalized deadline. There is little difference in the performance of the dynamic algorithms for the Montage application. This is because the exponential score metric hides the long tail of low-priority workflows that have successfully completed (e.g., completing a workflow with priority 10 adds 2^{-10} to the score). Figure 10 shows that the file locality-aware dynamic algorithms are able to complete more workflows within the same deadline. Knowing that the algorithms also have a slightly higher exponential score we conclude that the algorithm produces better schedules. One notable case where this is not the case is the CyberShake application with larger deadlines (Fig. 9). Here the simplest, unmodified WA-DPDS algorithm performs best. This is because the workflow runtimes are overestimated in the presence of a cache. Therefore, WA-DPLS rejects workflows from execution when they could actually finish within budget and deadline. When the VM cache is disabled, however, Storage-Aware algorithms always have the best scores, as Fig. 11 shows.

The static storage-aware algorithm outperforms its unaware counterpart. It is superior both in terms of score and the number of completed workflows. For the Montage application it sometimes performs worse. This is, again, caused by overestimated transfer times.

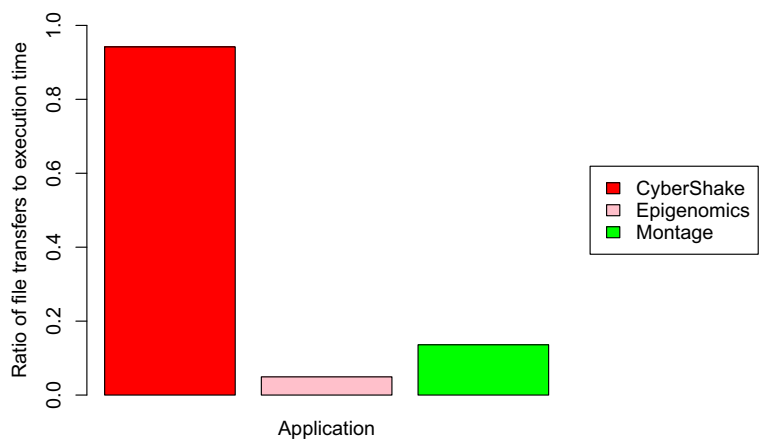

Fig. 12 Average ratio of total time spent on transfers to total ensemble execution time, per application for distributed storage system

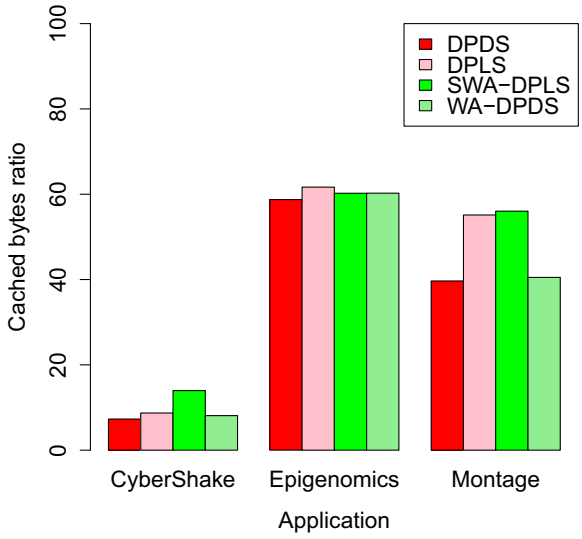

Fig. 13 Average cache bytes hit ratio per application for distributed storage system

Investigating how to improve the estimation function is a non-trivial task, because it requires simulating the state of the entire execution environment. Potential improvements have been left for future work.

Figure 12 shows the ratio of file transfer time to the total ensemble execution time averaged for all algorithms for this experiment. The CyberShake application spends the vast majority (94 %) of its total execution time on file transfers. This is caused by large input files of root tasks that have to be staged in to VMs. In comparison, Montage and Epigenomics spend 14 % and 5 % of their time on transfers.

Finally, Fig. 13 shows the average cache bytes hit ratio per application and algorithm. This value is defined as the ratio of the total number of bytes retrieved from the cache to the total size of all files requested to be transferred. One thing to note is that even unmodified algorithms produce schedules with significant cache hit ratios. Some applications, such as Epigenomics, are naturally cache-friendly and have high cache hit ratios regardless of schedule. This is caused by the fact that many of the tasks in these workflows share the same input files. The increase in hit ratio for the file locality-aware algorithms varies by application. It is as low as 1.4 percentage points for CyberShake with DPDS and DPLS algorithms, and as high as 15.5 percentage points for Montage with WA-DPDS and SWA-DPLS algorithms. This confirms that file locality-aware scheduling procedures can reduce the time spent on file transfers for certain application types.

6.4 NFS Storage

Our final experiment was designed to simulate an NFS server that is connected to all VMs. The NFS share is backed by an array of disks in a RAID 5 configuration [7]. The replica count was set to 5, the maximum read and write bandwidths were 20MiB/s, the latency was 200 ms, and the local VM cache size was 50GiB. The experiment consisted of running 2,100 simulations for CyberShake, Montage and SIPHT. Table 3 shows the average exponential score for each pair of application and algorithm, averaged over all simulation runs. For comparison, the table includes the results for the environment, where storage is infinitely fast, from Section 6.1. The results for NFS storage are similar to what we presented in previous sections in that the storage and file locality-aware algorithms outperform their original variants.

When comparing performance metrics for algorithms with storage disabled and NFS storage we can see that for certain applications the average score

Table 3 Average exponential score of all application and algorithm pairs in the NFS server storage experiment (**A**), compared to environment where storage is infinitely fast (**B**)

		DPDS	DPLS	SA-SPSS	SPSS	SWA-DPDS	WA-DPDS	SWA-DPLS
A	CyberShake	0.2388	0.2563	0.2593	0.0000	0.3756	0.2638	0.3810
	Montage	0.6224	0.6282	0.3369	0.1321	0.6981	0.6640	0.7083
	SIPHT	1.0315	1.0371	1.1791	0.2810	1.1390	1.1026	1.1443
B	CyberShake	1.1117	1.1111	0.8983	0.8983	1.1799	1.1799	1.1780
	Montage	0.8664	0.8697	0.7513	0.7513	0.9182	0.9182	0.9232
	SIPHT	1.0925	1.1617	1.2202	1.2202	1.1655	1.1655	1.1947

is substantially lower for all algorithms when storage is enabled. This is particularly visible for the data-intensive applications. For example, the score for DPLS and CyberShake is reduced from 1.1111 to 0.2563 or from 0.7513 to 0.1321 for SPSS and Montage. This is an important observation, because it shows that file operations may have substantial effect on ensemble execution. As a result, global storage may become a bottleneck for workflow execution when it is backed by slow resources (e.g., NFS storage).

7 Conclusions and Future Work

As the popularity of running scientific workflow applications in the cloud grows, it is important to optimize the performance of ensemble scheduling algorithms in such environments. In this paper we propose a model for simulating various storage systems that can be used on IaaS clouds. We propose modifications to our original algorithms to take into account data transfers. Most importantly, we propose and evaluate a novel dynamic scheduling algorithm that is aware of the underlying storage system. This algorithm optimizes schedules by taking advantage of file locality to reduce the number of file transfers during execution.

We show that for most data-intensive applications the locality-aware algorithms perform better than other algorithms we evaluated. It is able to successfully complete more higher priority workflows within the budget and deadline constraints. We observe that the slower the global storage is the better the locality aware algorithm performs compared to other ones. For example, the difference in the exponential score between DPLS and DPDS for the CyberShake application is higher in the distributed storage scenario than in the in-memory scenario. We also conclude that our modifications to the original algorithms are essential. Without them, the performance of the static algorithm (SPSS) is prohibitively low. Also, the workflow admission procedure used in WA-DPDS performs worse without our modifications.

The local VM cache has a low-to-medium impact on workflow execution, depending on application characteristics and what storage system is used. Some applications perform similarly in scenarios with the cache enabled and disabled. This is because their tasks have unique input files (e.g., CyberShake) or, alternatively, most of the execution time is spent on computations (e.g., Epigenomics). Some applications, however, perform better when the local VM cache is enabled. This is usually true for applications like Montage, which have large cache hit ratios and spend a noticeable part of execution time on data transfers.

We evaluate scientific applications in the context of how file transfers affect their execution. We show that some applications may spend as much as 90 % of their execution time on file transfers when operating on top of a distributed storage system, such as Google Cloud Storage. What is more, we observe that caching files in local VM storage is of great significance. Some applications exhibit cache hit ratios greater than 50 %.

Improvements to the file transfer estimation procedure are left for future work. We acknowledge that the procedure often overestimates transfer times, which creates room for improvement. Additionally, we reason that the SPSS algorithm is particularly vulnerable to dynamic changes in the environment that result in missed deadlines. This may be improved by introducing safety margins in the algorithm. Introducing a billing model for storage is also a possible future research direction.

The results of this research can be used by scientists running their applications on clouds to determine the performance of ensemble scheduling algorithms. Additionally, the impact of the chosen storage system on workflow execution can be assessed. This information may be used to evaluate storage systems and workflow applications themselves.

The work presented here can be extended to the scope of heterogeneous or hybrid clouds. We plan to address storage and data placement problems in multi-cloud environments. We consider this to be a global optimization problem that is different from non-trivial challenge of selecting appropriate storage systems and scheduling workflows within a homogeneous cloud. In the future we aim to combine these local and global approaches, using the methods presented in this paper and in our related research [38].

Based on our earlier experiments on Amazon EC2 [5] and on the current results, we are now in the process of implementing an Ensemble Manager. It will be available in a future release of Pegasus and will contribute to our research agenda.

Acknowledgments This work was funded by the National Science Foundation under the ACI SI2-SSI program grant #1148515. We thankfully acknowledge the support of the EU FP7-ICT project PaaSage (317715), Polish grant 3033/7PR/2014/2 and AGH grant 11.11.230.124.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Cloud workflow simulator project: https://github.com/malawski/cloudworkflowsimulator, accessed: 2015-05-02
2. Google cloud pricing: https://cloud.google.com/pricing/, accessed: 2015-05-02
3. Google cloud storage: https://cloud.google.com/storage/, accessed: 2015-05-01
4. Google compute engine: https://cloud.google.com/compute/, accessed: 2015-09-09
5. Agarwal, R., Juve, G., Deelman, E.: Peer-to-peer data sharing for scientific workflows on amazon ec2. In: Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis, SC Companion. 82–89. IEEE (2012)
6. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. Transactions on Parallel and Distributed Systems **25**(3), 682–694 (2014)
7. Arpaci-Dusseau, R.H., Arpaci-Dusseau, A.C.: Operating systems: Three easy pieces. Arpaci-Dusseau Books (2014)
8. Berriman, G.B., Deelman, E., Juve, G., Rynge, M., Vöckler, J.S.: The application of cloud computing to scientific workflows: a study of cost and performance. Philosophical Transactions of the Royal Society A: Mathematical. Phys. Eng. Sci **371**(1983), 20120066 (2013)
9. Berriman, G.B., Juve, G., Deelman, E., Regelson, M., Plavchan, P.: The application of cloud computing to astronomy: A study of cost and performance. In: Proceedings of 6th International Conference on e-Science, Workshops. 1–7. IEEE (2010)
10. Bharathi, S., Chervenak, A.: Data staging strategies and their impact on the execution of scientific workflows. In: Proceedings of the second international workshop on Data-aware distributed computing. ACM (2009)
11. Bharathi, S., Chervenak, A.: Scheduling data-intensive workflows on storage constrained resources. In: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science - WORKS '09. pp. 1–10. ACM Press, New York, New York, USA. http://dl.acm.org/citation.cfm?id=1645164.1645167 (2009)
12. Bittencourt, L.F., Sakellariou, R., Madeira, E.R.: Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In: Proceedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). 27–34 (2010)
13. Bittencourt, L.F., Madeira, E.R.M.: Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. J. Int. Se. Appl. **2**(3), 207–227 (2011)
14. Bocchi, E., Mellia, M., Sarni, S.: Cloud storage service benchmarking: Methodologies and experimentations. In: Cloud Networking, 2014 et al. 3rd International Conference on. 395–400 (2014)
15. Callaghan, S., Maechling, P., Small, P., Milner, K., Juve, G., Jordan, T.H., Deelman, E., Mehta, G., Vahi, K., Gunter, D., et al.: Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. International Journal of High Performance Computing Applications **25**(3), 274–285 (2011)
16. Çatalyürek, U.V., Kaya, K., Uçar, B.: Integrated data placement and task assignment for scientific workflows in clouds. http://portal.acm.org/citation.cfm?id=1996014.1996022 (2011)
17. Chen, W., Deelman, E.: Partitioning and scheduling workflows across multiple sites with storage constraints. In: Parallel Processing and Applied Mathematics, LNCS 7204, 11–20. Springer (2012)
18. Chiang, R.C., Huang, H.H.: TRACON. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis - SC '11. p. 1. ACM Press, New York, New York, USA. http://dl.acm.org/citation.cfm?id=2063384.2063447 (2011)
19. Costa, L.B., Yang, H., Vairavanathan, E., Barros, A., Maheshwari, K., Fedak, G., Katz, D., Wilde, M., Ripeanu, M., Al-Kiswany, S.: The Case for Workflow-Aware Storage: An Opportunity Study. Journal of Grid Computing **13**(1) (2014). doi:10.1007/s10723-014-9307-6
20. Dan, A., Towsley, D.: An approximate analysis of the lru and fifo buffer replacement schemes. SIGMETRICS Perform. Eval. Rev. **18**(1), 143–152 (1990). doi:10.1145/98460.98525
21. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., et al: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Sci. Program **13**(3), 219–237 (2005)
22. Duan, R., Prodan, R., Li, X.: Multi-objective game theoretic scheduling of bag-of-tasks workflows on hybrid clouds. Transactions on Cloud Computing **2** (1), 29–42 (2014)
23. Fitzpatrick, B.: Distributed caching with memcached. Linux journal 2004 **5**(124) (2004)
24. Garcia-Molina, H., Salem, K.: Main memory database systems: An overview. IEEE Trans. Knowl. Data Eng. **4**(6), 509–516 (1992)
25. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: ACM SIGOPS Operating Systems Review. vol. 37, 29–43. ACM (2003)
26. Graves, R., Jordan, T.H., Callaghan, S., Deelman, E., Field, E., Juve, G., Kesselman, C., Maechling, P., Mehta, G., Milner, K.E.A.: Cybershake: A physics-based seismic hazard model for southern california. Pure Appl. Geophys. **168**(3-4), 367–381 (2011)

27. Gunarathne, T., Zhang, B., Wu, T.L., Qiu, J.: Scalable parallel computing on clouds using Twister4Azure iterative MapReduce. Futur. Gener. Comput. Syst. **29**(4), 1035–1048 (2013). http://www.sciencedirect.com/science/article/pii/S0167739X12001379
28. Hill, Z., Li, J., Mao, M., Ruiz-Alvarez, A., Humphrey, M.: Early observations on the performance of windows azure. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 367–376. ACM (2010)
29. Hoenisch, P., Hochreiner, C., Schuller, D., Schulte, S., Mendling, J., Dustdar, S.: Cost-efficient scheduling of elastic processes in hybrid clouds. In: Proceedings of 8th International Conference on Cloud Computing. 17–24. IEEE (2015). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7214023
30. Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J.C., Laity, A., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Prince, T., et al.: Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. Int. J. Comput. Sci. Eng. **4**(2), 73–87 (2009)
31. Jellinek, R., Zhai, Y., Ristenpart, T., Swift, M.: A day late and a dollar short: the case for research on cloud billing systems. In: Proceedings of the 6th USENIX conference on Hot Topics in Cloud Computing. pp. 21–21. USENIX Association (2014)
32. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Futur. Gener. Comput. Syst **29**(3), 682–692 (2013)
33. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B.P., Maechling, P.: Data sharing options for scientific workflows on amazon ec2. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–9. IEEE Computer Society (2010)
34. Kaur, G., Moghariya, U., Reed, J.: Understanding and taming the variability of cloud storage latency. Tech. rep. (2013)
35. Kosar, T., Balman, M.: A new paradigm: Data-aware scheduling in grid computing. Futur. Gener. Comput. Syst. **25**(4), 406–413 (2009)
36. Livny, J., Teonadi, H., Livny, M., Waldor, M.K.: High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. PloS one **3**(9), e3197 (2008)
37. Maechling, P.e.a.: Scec cybershake workflows automating probabilistic seismic hazard analysis calculations. In: Workflows for e-Science, 143–163. Springer (2007)
38. Malawski, M., Figiela, K., Bubak, M., Deelman, E., Nabrzyski, J.: Scheduling multi-level deadline-constrained scientific workflows on clouds based on cost optimization. Scientific Programming (2015)
39. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Futur. Gener. Comput. Syst. **48**, 1–18 (2015)
40. Mao, M., Humphrey, M.: A performance study on the vm startup time in the cloud. In: Proceedings of the 5th International Conference on Cloud Computing. pp. 423–430. IEEE (2012)
41. Pereira, W.F., Bittencourt, L.F., da Fonseca, N.L.S.: Scheduler for data-intensive workflows in public clouds. In: Proceedings of 2nd Latin American Conference on Cloud Computing and Communications. 41–46. IEEE. http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6842221 (2013)
42. Ramakrishnan, A., Singh, G., Zhao, H., Deelman, E., Sakellariou, R., Vahi, K., Blackburn, K., Meyers, D., Samidi, M.: Scheduling data-intensive workflows onto storage-constrained distributed resources. In: Proceedings of 7th International Symposium on Cluster and Grid Computing (CCGrid). pp. 401–409. IEEE (2007)
43. Ranganathan, K., Foster, I.: Simulation studies of computation and data scheduling algorithms for data grids. Journal of Grid computing **1**(1), 53–62 (2003)
44. Rodriguez, M., Buyya, R.: Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. Transactions on Cloud Computing **2**(2), 222–235 (2014)
45. Schmuck, F.B., Haskin, R.L.: Gpfs: A shared-disk file system for large computing clusters. In: FAST. vol. 2, 19 (2002)
46. da Silva, R.F., Chen, W., Juve, G., Vahi, K., Deelman, E.: Community resources for enabling research in distributed scientific workflows. In: Proceedings of 10th International Conference one-Science. vol. 1, 177–184. IEEE (2014)
47. Tolosana-Calasanz, R., BañAres, J.Á., Pham, C., Rana, O.F.: Enforcing qos in scientific workflow systems enacted over cloud infrastructures. J. Comput. Syst. Sci. **78**(5), 1300–1315 (2012)
48. Topcuoglu, H., Hariri, S., Wu, M.y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. Transactions on Parallel and Distributed Systems **13**(3), 260–274 (2002)
49. Vöckler, J.S., Juve, G., Deelman, E., Rynge, M., Berriman, B.: Experiences using cloud computing for a scientific workflow application. In: Proceedings of the 2nd international workshop on Scientific cloud computing. pp. 15–24. ACM (2011)
50. Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M.: Characterizing facebook's memcached workload. IEEE Internet Computing **18**(2), 41–49 (2014)
51. Yuan, D., Yang, Y., Liu, X., Chen, J.: A cost-effective strategy for intermediate data storage in scientific cloud workflow systems. In: Proceedings of International Parallel and Distributed Processing Symposium (IPDPS). pp. 1–12. IEEE, Atlanta, GA, USA. doi:10.1109/ipdps.2010.5470453 (2010)
52. Zhang, S., Zhang, S., Chen, X., Huo, X.: Cloud computing research and development trend. In: Proceedings of Second International Conference on Future Networks, ICFN'10. pp. 93–97. IEEE (2010)
53. Zhang, Z., Katz, D., Wilde, M., Wozniak, J., Foster, I.: MTC envelope: defining the capability of large scale computers in the context of parallel scripting applications. In: Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing - HPDC '13. pp. 37–48. ACM, New York, NY, USA. doi:10.1145/2462902.2462913 (2013)